Още итеравивни алгоритми

Тодор Дуков

Подход при задачи с вече даден алгоритъм

Нека разгледаме следния алгоритъм:

```
1  int foo(int a)
2  {
3      int x = 6, y = 1, z = 0;
4  
5      for (int i = 0; i < a; ++i)
6      {
7          z += y;
8          y += x;
9          x += 6;
10     }
11 
12     return z;
13 }
```

Питаме се какво връща той?
Обикновено в такива задачи трябва да се изпробва алгоритъма върху няколко стойности. Можем да забележим, че:

- `foo`(0) връща 0
- `foo`(1) връща 1
- `foo`(2) връща 8
- `foo`(3) връща 27
- `foo`(4) връща 64
- `foo`(5) връща 125
- `foo`(6) връща 216
- `foo`(7) връща 343
- `foo`(8) връща 512

Вече можем да видим какво прави алгоритъмът – `foo`(`a`) връща a^3. Нека сега докажем това:

Инвариант. *При всяко достигане на проверката за край на цикъла на ред* 5 *имаме, че:*

- $x = 6(1 + i)$
- $y = 3i^2 + 3i + 1$
- $z = i^3$

База. При първото достигане имаме, че:

- $i = 0$
- $x = 6 = 6 \cdot 1 = 6(1 + 0) = 6(1 + i)$
- $y = 1 = 3 \cdot 0^2 + 3 \cdot 0 + 1 = 3i^2 + 3i + 1$
- $z = 0 = 0^3 = i^3$

Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава при влизане в тялото на цикъла:

- $\mathtt{z}$ ще стане $\mathtt{z} + \mathtt{y} \overset{(\text{ИП})}{=} \mathtt{i}^3 + 3\mathtt{i}^2 + 3\mathtt{i} + 1 = (\underbrace{\mathtt{i}+1}_{\text{ново } \mathtt{i}})^3$
- $\mathtt{y}$ ще стане $\mathtt{y} + \mathtt{x} \overset{(\text{ИП})}{=} 3\mathtt{i}^2 + 3\mathtt{i} + 1 + 6 + 6\mathtt{i} = 3(\underbrace{\mathtt{i}+1}_{\text{ново } \mathtt{i}})^2 + 3(\underbrace{\mathtt{i}+1}_{\text{ново } \mathtt{i}}) + 1$
- $\mathtt{x}$ ще стане $\mathtt{x} + 6 \overset{(\text{ИП})}{=} 6(1+\mathtt{i}) + 6 = 6(1 + \underbrace{\mathtt{i}+1}_{\text{ново } \mathtt{i}})$

Терминация. В последното достигане на проверката за край на цикъла имаме, че $\mathtt{i} = \mathtt{a}$, и тогава на ред 12 алгоритъмът ще върне $\mathtt{z} = \mathtt{a}^3$.
Нека разгледаме още един такъв пример:

```
1  int bar(int n)
2  {
3      return (int)sqrt((double)n * n);
4  }
5
6  int foo(int x, int y)
7  {
8      return (x + y + bar(x - y)) / 2;
9  }
10
11 int quux(int *arr, int n)
12 {
13     int a = arr[0];
14
15     for (int i = 1; i < n; ++i)
16     {
17         a = foo(a, arr[i]);
18     }
19
20     return a;
21 }
```

Искаме да видим какво връща $\mathtt{quux}(\mathtt{arr}, \mathtt{n})$ където $\mathtt{arr}$ е масив от цели числа а $\mathtt{n}$ е броят на неговите елементи. Тук най-трудното, което трябва да се направи, е да се определи какво връщат `bar` и `foo`:

- $\mathtt{bar}(\mathtt{n}) = \sqrt{\mathtt{n}^2} = |\mathtt{n}|$:
  - ако $\mathtt{n} \geq 0$, то $\sqrt{\mathtt{n}^2} = \mathtt{n} = |\mathtt{n}|$
  - ако $\mathtt{n} < 0$, то $\sqrt{\mathtt{n}^2} = -\mathtt{n} = |\mathtt{n}|$
- $\mathtt{foo}(\mathtt{x}, \mathtt{y}) = \frac{\mathtt{x}+\mathtt{y}+|\mathtt{x}-\mathtt{y}|}{2} = \max\{\mathtt{x}, \mathtt{y}\}$:
  - ако $\mathtt{x} \geq \mathtt{y}$, то $\frac{\mathtt{x}+\mathtt{y}+|\mathtt{x}-\mathtt{y}|}{2} = \frac{\mathtt{x}+\mathtt{y}+\mathtt{x}-\mathtt{y}}{2} = \frac{2\mathtt{x}}{2} = \mathtt{x} = \max\{\mathtt{x}, \mathtt{y}\}$
  - ако $\mathtt{x} < \mathtt{y}$, то $\frac{\mathtt{x}+\mathtt{y}+|\mathtt{x}-\mathtt{y}|}{2} = \frac{\mathtt{x}+\mathtt{y}+\mathtt{y}-\mathtt{x}}{2} = \frac{2\mathtt{y}}{2} = \mathtt{y} = \max\{\mathtt{x}, \mathtt{y}\}$

Като знаем какво правят `bar` и `foo`, можем да забележим, че $\mathtt{quux}(\mathtt{arr}, \mathtt{n})$ дава най-големия елемент на `arr`:

Инвариант. *При всяко достигане на проверката за край на цикъла на ред* 15 *имаме, че* $\mathtt{a} = \max \mathtt{arr}[0 \dots \mathtt{i} - 1]$.

База. При първото достигане имаме, че $\mathtt{a} = \mathtt{arr}[0] = \max \mathtt{arr}[0 \dots 1 - 1] = \max \mathtt{arr}[0 \dots \mathtt{i} - 1]$.
Поддръжка. Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава като влезем в тялото на цикъла, променливата $\mathtt{a}$ става:

$$\begin{aligned}\mathtt{foo}(\mathtt{a}, \mathtt{arr}[\mathtt{i}]) \overset{(\text{ИП})}{=} \mathtt{foo}(\max \mathtt{arr}[0 \dots \mathtt{i}-1], \mathtt{arr}[\mathtt{i}]) &= \max(\max \mathtt{arr}[0 \dots \mathtt{i}-1], \mathtt{arr}[\mathtt{i}]) \\ &= \max \mathtt{arr}[0 \dots \mathtt{i}] = \max \mathtt{arr}[\mathtt{i} \dots \underbrace{\mathtt{i}+1}_{\text{ново } \mathtt{i}} - 1]\end{aligned}$$

Терминация. При последното достигане на проверката за край на цикъла на ред 15 променливата $\mathtt{i}$ ще бъде $\mathtt{n}$, откъдето функцията ще върне точно $\mathtt{a} = \max \mathtt{arr}[0 \dots \mathtt{n} - 1]$, с което сме готови.

Задачи

Задача 1. Даден е следният алгоритъм:

```
int num_slopes(int *arr, int n)
{
    int slopes = 1;

    for (int i = 1; i < n; ++i)
    {
        if (arr[i - 1] > arr[i])
            ++slopes;
    }

    return slopes;
}
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, функцията `num_slopes(arr, n)` връща броят на максималните ненамаляващи подмасиви на `arr`.

Задача 2. Даден е следният алгоритъм:

```
int kadane(int *arr, int n)
{
    int max_so_far = -INF, max_ending_here = 0;

    for (int i = 0; i < n; i++)
    {
        max_so_far = max(arr[i], max_so_far + arr[i]);
        max_ending_here = max(max_ending_here, max_so_far);
    }

    return max_so_far;
}
```

Какво връща `kadane(arr, n)`, където `arr` е масив от цели числа с дължина `n`? Обосновете отговора си.

Задача 3. Даден е следният алгоритъм:

```
int find_majority(int *arr, int n)
{
    int maj = arr[0];
    int cnt = 1;

    for (int i = 1; i < n; ++i)
    {
        if (cnt == 0)
        {
            maj = arr[i];
            cnt = 1;
        }
        else
            cnt += (arr[i] == maj ? 1 : -1);
    }

    return maj;
}
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, в който има елемент с повече от $\lfloor \frac{\mathtt{n}}{2} \rfloor$ срещания, функцията `find_majority(arr, n)` ще върне точно този елемент.